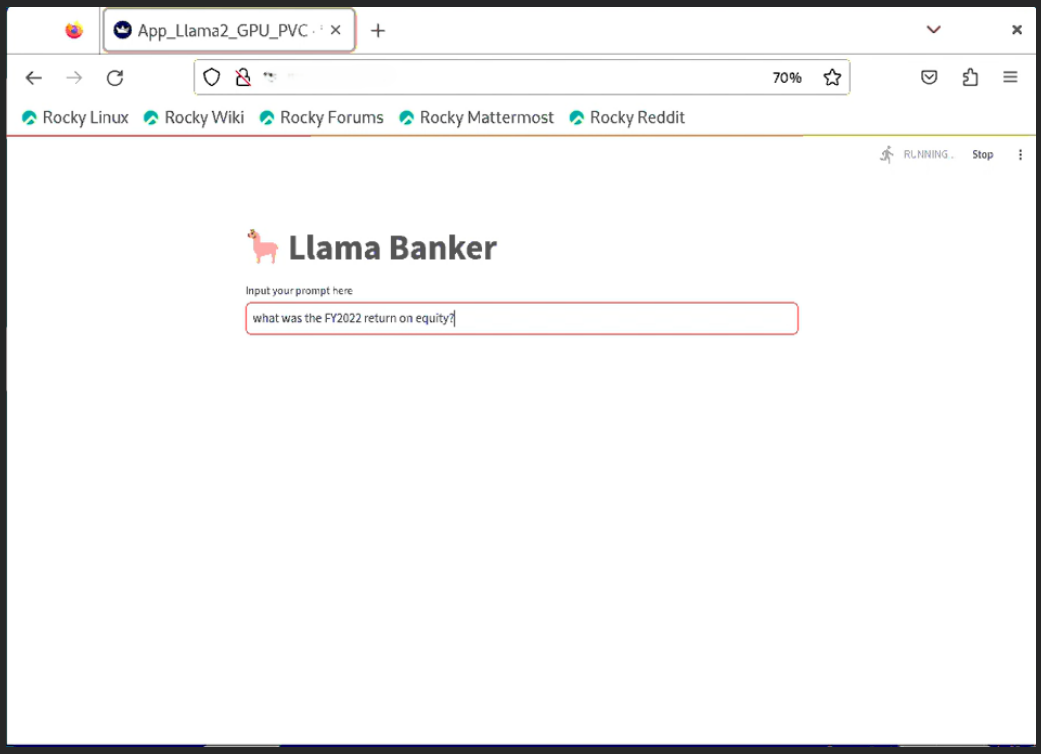
RAG on GPU Max



Introduction

In 2023, Intel launched its large-scale GPU, the Intel GPU Max, targeting high-performance computing (HPC) and AI applications at a cost-efficient price. Around the same time, OpenAI revolutionized the AI landscape with ChatGPT, reigniting the AI hype with generative AI (GenAI). As organizations began to adopt large language models (LLMs) like ChatGPT, they quickly recognized both the potential efficiency gains for employees and the risks associated with uncontrolled use. Publicly available LLMs can learn and retain information from users, posing a risk of inadvertently exposing confidential company data.

RAG to guardrail LLMs for enterprises

To address this concern, enterprises can deploy LLMs combined with Retrieval-Augmented Generation (RAG) on their own IT infrastructure, whether on-premises or in a cloud-based virtual data center. This approach ensures that the LLM is isolated from public use, safeguarding sensitive information. Additionally, RAG allows enterprises to provide secure, confidential information as a knowledge base to the LLM and establish guardrails without the need for expensive and resource-intensive model training or fine-tuning. Unlike training and fine-tuning, which require significant computational resources, RAG is less compute-intensive and more feasible for enterprises to implement.

Intel's GPU Max RAG

Since RAG is the lowest-hanging fruit to provide a safe and tailored LMMs use for enterprises, I built a RAG for Intel's GPU Max.
To do that , I adapted the code from Nicholas Renotte available here: https://github.com/nicknochnack/Llama2RAG
That RAG implementation is intended for Nvidia GPUs, so I modified it to run on Intel GPU Max. You can see my code and tutorial here: https://github.com/TheFavAI/Intel-GPU-Max-RAG
A common misconception is that you need to convert CUDA code, running on Nvidia GPUs, into DPC++, running on Intel GPUs. The thing is that it isn't the case. Most AI Engineer, Data scientist, etc. code in Python. And by using Python, they usually don't need to code a single line of CUDA or DPC++. Indeed, everything is handled by the libraries used in their python codes!
The adaptions I had to do to the code of Nicolas Renotte are minimal.

Changes to the code

Added line 5 :
import intel_extension_for_pytorch as ipex

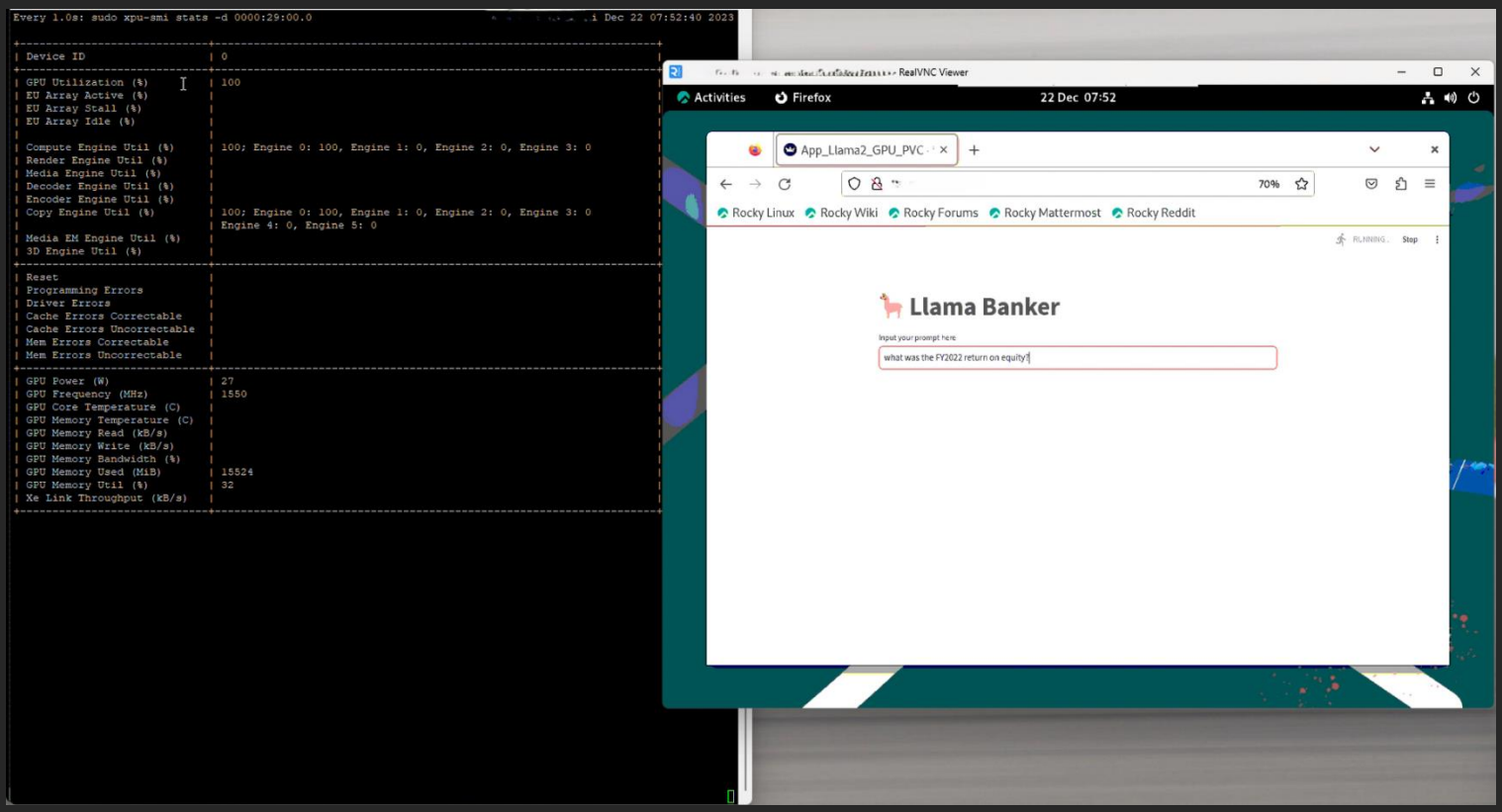
Deletion of “, load_in_8bit=True” and addition of “.to("xpu")” line 25-27:
model = AutoModelForCausalLM.from_pretrained(name, cache_dir='./model/', use_auth_token=auth_token, torch_dtype=torch.float16, rope_scaling={"type": "dynamic", "factor": 2}).to("xpu")

Added line 28:
model = ipex.optimize(model)

Change to the pdf loader functions, but it has nothing to do with the RAG itself:
file_path_str = str(Path('./data/annualreport.pdf'))
documents = loader.load(file_path=file_path_str, metadata=True)

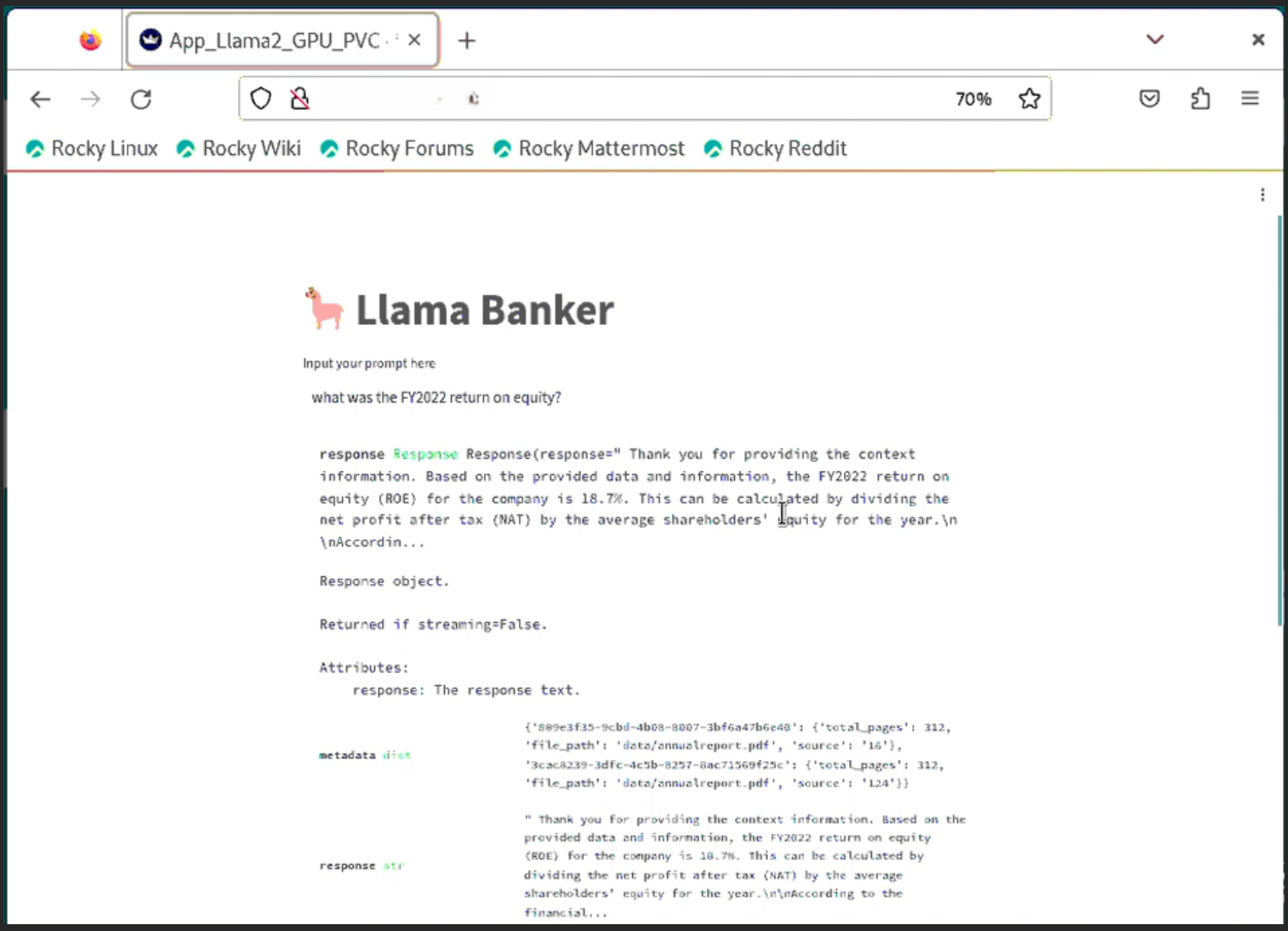
How does it look?

Here is the code running. You can see the GPU being at 100% utilization to provide the answer as fast as possible.



The result

Here is an example of the answer the RAG can provider. The input is a company’s financials, a huge pdf of more than 350 pages with a lot of numbers and complex financial notions. I asked an industry specific question to the RAG. And it manages to successfully answer as you can see.



Next steps

- As you can see, the UI isn't as good as it could be.
- The RAG only takes 1 document as an input, even though the document is huge. The RAG would need a database to handle multiple documents.